Chapter 13

Mining Text Data

"*The first forty years of life give us the text; the next thirty supply the commentary on it.*"—Arthur Schopenhauer

13.1 Introduction

Text data are copiously found in many domains, such as the Web, social networks, newswire services, and libraries. With the increasing ease in archival of human speech and expression, the volume of text data will only increase over time. This trend is reinforced by the increasing digitization of libraries and the ubiquity of the Web and social networks. Some examples of relevant domains are as follows:

1. *Digital libraries:* A recent trend in article and book production is to rely on digitized versions, rather than hard copies. This has led to the proliferation of digital libraries in which effective document management becomes crucial. Furthermore mining tools are also used in some domains, such as biomedical literature, to glean useful insights.

2. *Web and Web-enabled applications:* The Web is a vast repository of documents that is further enriched with links and other types of side information. Web documents are also referred to as *hypertext*. The additional side information available with hypertext can be useful in the knowledge discovery process. In addition, many web-enabled applications, such as social networks, chat boards, and bulletin boards, are a significant source of text for analysis.

3. *Newswire services:* An increasing trend in recent years has been the de-emphasis of printed newspapers and a move toward electronic news dissemination. This trend creates a massive stream of news documents that can be analyzed for important events and insights.

The set of features (or dimensions) of text is also referred to as its *lexicon*. A collection of documents is referred to as a *corpus*. A document can be viewed as either a sequence, or a multidimensional record. A text document is, after all, a discrete sequence of words, also

C. C. Aggarwal, *Data Mining: The Textbook*, DOI 10.1007/978-3-319-14142-8_13

referred to as a *string*. Therefore, many sequence-mining methods discussed in Chap. 15 are theoretically applicable to text. However, such sequence mining methods are rarely used in the text domain. This is partially because sequence mining methods are most effective when the length of the sequences and the number of possible tokens are both relatively modest. On the other hand, documents can often be long sequences drawn on a lexicon of several hundred thousand words.

In practice, text is usually represented as multidimensional data in the form of frequency-annotated bag-of-words. Words are also referred to as *terms*. Although such a representation loses the ordering information among the words, it also enables the use of much larger classes of multidimensional techniques. Typically, a preprocessing approach is applied in which the very common words are removed, and the variations of the same word are consolidated. The processed documents are then represented as an *unordered* set of words, where normalized frequencies are associated with the individual words. The resulting representation is also referred to as the *vector space representation* of text. The vector space representation of a document is a multidimensional vector that contains a frequency associated with each word (dimension) in the document. The overall dimensionality of this data set is equal to the number of distinct words in the lexicon. The words from the lexicon that are not present in the document are assigned a frequency of 0. Therefore, text is not very different from the multidimensional data type that has been studied in the preceding chapters.

Due to the multidimensional nature of the text, the techniques studied in the aforementioned chapters can also be applied to the text domain with a modest number of modifications. What are these modifications, and why are they needed? To understand these modifications, one needs to understand a number of specific characteristics that are unique to text data:

1. *Number of "zero" attributes:* Although the base dimensionality of text data may be of the order of several hundred thousand words, a single document may contain only a few hundred words. If each word in the lexicon is viewed as an attribute, and the document word frequency is viewed as the attribute value, most attribute values are 0. This phenomenon is referred to as high-dimensional *sparsity*. There may also be a wide variation in the number of nonzero values across different documents. This has numerous implications for many fundamental aspects of text mining, such as distance computation. For example, while it is possible, in theory, to use the Euclidean function for measuring distances, the results are usually not very effective from a practical perspective. This is because Euclidean distances are extremely sensitive to the varying document lengths (the number of nonzero attributes). The Euclidean distance function cannot compute the distance between two short documents in a comparable way to that between two long documents because the latter will usually be larger.

2. *Nonnegativity:* The frequencies of words take on nonnegative values. When combined with high-dimensional sparsity, the nonnegativity property enables the use of specialized methods for document analysis. In general, all data mining algorithms must be cognizant of the fact that the presence of a word in a document is statistically more significant than its absence. Unlike traditional multidimensional techniques, incorporating the *global* statistical characteristics of the data set in pairwise distance computation is crucial for good distance function design.

3. *Side information:* In some domains, such as the Web, additional side information is available. Examples include hyperlinks or other metadata associated with the document. These additional attributes can be leveraged to enhance the mining process further.

This chapter will discuss the adaptation of many conventional data mining techniques to the text domain. Issues related to document preprocessing will also be discussed.

This chapter is organized as follows. Section 13.2 discusses the problem of document preparation and similarity computation. Clustering methods are discussed in Sect. 13.3. Topic modeling algorithms are addressed in Sect. 13.4. Classification methods are discussed in Sect. 13.5. The first story detection problem is discussed in Sect. 13.6. The summary is presented in Sect. 13.7.

13.2 Document Preparation and Similarity Computation

As the text is not directly available in a multidimensional representation, the first step is to convert raw text documents to the multidimensional format. In cases where the documents are retrieved from the Web, additional steps are needed. This section will discuss these different steps.

1. *Stop word removal:* Stop words are frequently occurring words in a language that are not very discriminative for mining applications. For example, the words "*a*," "*an*," and "*the*" are commonly occurring words that provide very little information about the actual content of the document. Typically, articles, prepositions, and conjunctions are stop words. Pronouns are also sometimes considered stop words. Standardized stop word lists are available in different languages for text mining. The key is to understand that almost all documents will contain these words, and they are usually not indicative of topical or semantic content. Therefore, such words add to the noise in the analysis, and it is prudent to remove them.

2. *Stemming:* Variations of the same word need to be consolidated. For example, singular and plural representations of the same word, and different tenses of the same word are consolidated. In many cases, stemming refers to common root extraction from words, and the extracted root may not even be a word in of itself. For example, the common root of *hoping* and *hope* is *hop*. Of course, the drawback is that the word *hop* has a different meaning and usage of its own. Therefore, while stemming usually improves *recall* in document retrieval, it can sometimes worsen *precision* slightly. Nevertheless, stemming usually enables higher quality results in mining applications.

3. *Punctuation marks:* After stemming has been performed, punctuation marks, such as commas and semicolons, are removed. Furthermore, numeric digits are removed. Hyphens are removed, if the removal results in distinct and meaningful words. Typically, a base dictionary may be available for these operations. Furthermore, the distinct parts of the hyphenated word can either be treated as separate words, or they may be merged into a single word.

After the aforementioned steps, the resulting document may contain only semantically relevant words. This document is treated as a bag-of-words, in which relative ordering is irrelevant. In spite of the obvious loss of ordering information in this representation, the bag-of-words representation is surprisingly effective.

13.2.1 Document Normalization and Similarity Computation

The problem of document normalization is closely related to that of similarity computation. While the issue of text similarity is discussed in Chap. 3, it is also discussed here for completeness. Two primary types of normalization are applied to documents:

1. *Inverse document frequency:* Higher frequency words tend to contribute noise to data mining operations such as similarity computation. The removal of stop words is motivated by this aspect. The concept of inverse document frequency generalizes this principle in a softer way, where words with higher frequency are weighted less.

2. *Frequency damping:* The repeated presence of a word in a document will typically bias the similarity computation significantly. To provide greater stability to the similarity computation, a damping function is applied to word frequencies so that the frequencies of different words become more similar to one another. It should be pointed out that frequency damping is optional, and the effects vary with the application at hand. Some applications, such as clustering, have shown comparable or better performance without damping. This is particularly true if the underlying data sets are relatively clean and have few spam documents.

In the following, these different types of normalization will be discussed. The inverse document frequency id_i of the ith term is a decreasing function of the number of documents n_i in which it occurs:

$$id_i = \log(n/n_i). \tag{13.1}$$

Here, the number of documents in the collection is denoted by n. Other ways of computing the inverse document frequency are possible, though the impact on the similarity function is usually limited.

Next, the concept of frequency damping is discussed. This normalization ensures that the excessive presence of a single word does not throw off the similarity computation. Consider a document with word-frequency vector $\overline{X} = (x_1 \ldots x_d)$, where d is the size of the lexicon. A damping function $f(\cdot)$, such as the square root or the logarithm, is optionally applied to the frequencies before similarity computation:

$$f(x_i) = \sqrt{x_i}$$
$$f(x_i) = \log(x_i).$$

Frequency damping is optional and is often omitted. This is equivalent to setting $f(x_i)$ to x_i. The *normalized* frequency $h(x_i)$ for the ith word may be defined as follows:

$$h(x_i) = f(x_i) id_i. \tag{13.2}$$

This model is popularly referred to as the tf-idf model, where tf represents the term frequency and idf represents the inverse document frequency.

The normalized representation of the document is used for data mining algorithms. A popularly used measure is the cosine measure. The cosine measure between two documents with raw frequencies $\overline{X} = (x_1 \ldots x_d)$ and $\overline{Y} = (y_1 \ldots y_d)$ is defined using their normalized representations:

$$\cos(\overline{X}, \overline{Y}) = \frac{\sum_{i=1}^{d} h(x_i)h(y_i)}{\sqrt{\sum_{i=1}^{d} h(x_i)^2}\sqrt{\sum_{i=1}^{d} h(y_i)^2}} \tag{13.3}$$

Another measure that is less commonly used for text is the *Jaccard coefficient* $J(\overline{X}, \overline{Y})$:

$$J(\overline{X}, \overline{Y}) = \frac{\sum_{i=1}^{d} h(x_i)h(y_i)}{\sum_{i=1}^{d} h(x_i)^2 + \sum_{i=1}^{d} h(y_i)^2 - \sum_{i=1}^{d} h(x_i)h(y_i)}. \tag{13.4}$$

The Jaccard coefficient is rarely used for the text domain, but it is used very commonly for sparse binary data as well as sets. Many forms of transaction and market basket data use the Jaccard coefficient. It needs to be pointed out that the transaction and market basket data share many similarities with text because of their sparse and nonnegative characteristics. Most text mining techniques discussed in this chapter can also be applied to these domains with minor modifications.

13.2.2 Specialized Preprocessing for Web Documents

Web documents require specialized preprocessing techniques because of some common properties of their structure, and the richness of the links inside them. Two major aspects of Web document preprocessing include the removal of specific parts of the documents (e.g., tags) that are not useful, and the leveraging of the actual structure of the document. HTML tags are generally removed by most preprocessing techniques.

HTML documents have numerous fields in them, such as the title, the metadata, and the body of the document. Typically, analytical algorithms treat these fields with different levels of importance, and therefore weigh them differently. For example, the title of a document is considered more important than the body and is weighted more heavily. Another example is the anchor text in Web documents. Anchor text contains a description of the Web page pointed to by a link. Due to its descriptive nature, it is considered important, but it is sometimes not relevant to the topic of the page itself. Therefore, it is often removed from the text of the document. In some cases, where possible, anchor text could even be added to the text of the document *to which it points*. This is because anchor text is often a summary description of the document to which it points.

A Web page may often be organized into content blocks that are not related to the primary subject matter of the page. A typical Web page will have many irrelevant blocks, such as advertisements, disclaimers, or notices, that are not very helpful for mining. It has been shown that the quality of mining results improve when only the text in the main block is used. However, the (automated) determination of main blocks from web-scale collections is itself a data mining problem of interest. While it is relatively easy to decompose the Web page into blocks, it is sometimes difficult to identify the main block. Most automated methods for determining main blocks rely on the fact that a *particular* site will typically utilize a similar layout for the documents on the site. Therefore, if a collection of documents is available from the site, two types of automated methods can be used:

1. *Block labeling as a classification problem:* The idea in this case is to create a new training data set that extracts visual rendering features for each block in the training data, using Web browsers such as Internet Explorer. Many browsers provide an API that can be used to extract the coordinates for each block. The main block is then manually labeled for some examples. This results in a training data set. The resulting training data set is used to build a classification model. This model is used to identify the main block in the remaining (unlabeled) documents of the site.

2. *Tree matching approach:* Most Web sites generate the documents using a fixed template. Therefore, if the template can be extracted, then the main block can be identified relatively easily. The first step is to extract *tag trees* from the HTML pages. These represent the frequent tree patterns in the Web site. The tree matching algorithm, discussed in the bibliographic section, can be used to determine such templates from these tag trees. After the templates have been found, it is determined, which block is the main one in the extracted template. Many of the peripheral blocks often have similar content in different pages and can therefore be eliminated.

13.3 Specialized Clustering Methods for Text

Most of the algorithms discussed in Chap. 6 can be extended to text data. This is because the vector space representation of text is also a multidimensional data point. The discussion in this chapter will first focus on generic modifications to multidimensional clustering algorithms, and then present specific algorithms in these contexts. Some of the clustering methods discussed in Chap. 6 are used more commonly than others in the text domain. Algorithms that leverage the nonnegative, sparse, and high-dimensional features of the text domain are usually preferable to those that do not. Many clustering algorithms require significant adjustments to address the special structure of text data. In the following, the required modifications to some of the important algorithms will be discussed in detail.

13.3.1 Representative-Based Algorithms

These correspond to the family of algorithms such as k-means, k-modes, and k-median algorithms. Among these, the k-means algorithms are the most popularly used for text data. Two major modifications are required for effectively applying these algorithms to text data.

1. The first modification is the choice of the similarity function. Instead of the Euclidean distance, the cosine similarity function is used.

2. Modifications are made to the computation of the cluster centroid. All words in the centroid are not retained. The low-frequency words in the cluster are projected out. Typically, a maximum of 200 to 400 words in each centroid are retained. This is also referred to as a *cluster digest*, and it provides a representative set of *topical words* for the cluster. Projection-based document clustering has been shown to have significant effectiveness advantages. A smaller number of words in the centroid speeds up the similarity computations as well.

A specialized variation of the k-means for text, which uses concepts from hierarchical clustering, will be discussed in this section. Hierarchical methods can be generalized easily to text because they are based on generic notions of similarity and distances. Furthermore, combining them with the k-means algorithm results in both stability and efficiency.

13.3.1.1 Scatter/Gather Approach

Strictly speaking, the *scatter/gather* terminology does not refer to the clustering algorithm itself but the browsing ability enabled by the clustering. This section will, however, focus on the clustering algorithm. This algorithm uses a combination of k-means clustering and hierarchical partitioning. While hierarchical partitioning algorithms are very robust, they typically scale worse than $\Omega(n^2)$, where n is the number of documents in the collection. On the other hand, the k-means algorithm scales as $O(k \cdot n)$, where k is the number of clusters. While the k-means algorithm is more efficient, it can sometimes be sensitive to the choice of seeds. This is particularly true for text data in which each document contains only a small part of the lexicon. For example, consider the case where the document set is to be partitioned into five clusters. A vanilla k-means algorithm will select five documents from the original data as the initial seeds. The number of distinct words in these five documents will typically be a *very small* subset of the entire lexicon. Therefore, the first few iterations of k-means may not be able to assign many documents meaningfully to clusters when they do not contain a significant number of words from this small lexicon subset. This initial

incoherence can sometimes be inherited by later iterations, as a result of which the quality of the final results will be poor.

To address this issue, the *scatter/gather* approach uses a combination of hierarchical partitioning and k-means clustering in a two-phase approach. An efficient and simplified form of hierarchical clustering is applied to a sample of the corpus, to yield a robust set of seeds in the first phase. This is achieved by using either of two possible procedures that are referred to as *buckshot* and *fractionation*, respectively. Both these procedures are different types of hierarchical procedures. In the second phase, the robust seeds generated in the first phase are used as the starting point of a k-means algorithm, as adapted to text data. The size of the sample in the first phase is carefully selected to balance the time required by the first phase and the second phase. Thus, the overall approach may be described as follows:

1. Apply either the *buckshot* or *fractionation* procedures to create a robust set of initial seeds.

2. Apply a k-means approach on the resulting set of seeds to generate the final clusters. Additional refinements may be used to improve the clustering quality.

Next, the *buckshot* and *fractionation* procedures will be described. These are two alternatives for the first phase with a similar running time. The fractionation method is the more robust one, but the buckshot method is faster in many practical settings.

- *Buckshot:* Let k be the number of clusters to be found and n be the number of documents in the corpus. The buckshot method selects a seed superset of size $\sqrt{k \cdot n}$ and then agglomerates them to k seeds. Straightforward agglomerative hierarchical clustering algorithms (requiring[1] quadratic time) are applied to this initial sample of $\sqrt{k \cdot n}$ seeds. As we use quadratically scalable algorithms in this phase, this approach requires $O(k \cdot n)$ time. This seed set is more robust than a naive data sample of k seeds because it represents the summarization of a larger sample of the corpus.

- *Fractionation:* Unlike the buckshot method, which uses a sample of $\sqrt{k \cdot n}$ documents, the fractionation method works with all the documents in the corpus. The fractionation algorithm initially breaks up the corpus into n/m buckets, each of size $m > k$ documents. An agglomerative algorithm is applied to each of these buckets to reduce them by a factor $\nu \in (0, 1)$. This step creates $\nu \cdot m$ agglomerated documents in each bucket, and therefore $\nu \cdot n$ agglomerated documents over all buckets. An "agglomerated document" is defined as the concatenation of the documents in a cluster. The process is repeated by treating each of these agglomerated documents as a single document. The approach terminates when a total of k seeds remains.

  It remains to be explained how the documents are partitioned into buckets. One possibility is to use a random partitioning of the documents. However, a more carefully designed procedure can achieve more effective results. One such procedure is to sort the documents by the index of the jth most common word in the document. Here, j is chosen to be a small number, such as 3, that corresponds to medium frequency words in the documents. Contiguous groups of m documents in this sort order are mapped to clusters. This approach ensures that the resulting groups have at least a few common words in them and are therefore not completely random. This can sometimes help in improving the quality of the centers.

[1] As discussed in Chap. 6, standard agglomerative algorithms require more than quadratic time, though some simpler variants of single-linkage clustering [469] can be implemented in approximately quadratic time.

The agglomerative clustering of m documents in the first iteration of the fractionation algorithm requires $O(m^2)$ time for each group, and sums to $O(n \cdot m)$ over the n/m different groups. As the number of individuals reduces geometrically by a factor of ν in each iteration, the total running time over all iterations is $O(n \cdot m \cdot (1+\nu+\nu^2+\ldots))$. For $\nu < 1$, the running time over all iterations is still $O(n \cdot m)$. By selecting $m = O(k)$, one still ensure a running time of $O(n \cdot k)$ for the initialization procedure.

The *buckshot* and *fractionation* procedures require $O(k \cdot n)$ time. This is equivalent to the running time of a single iteration of the k-means algorithm. As discussed below, this is important in (asymptotically) balancing the running time of the two phases of the algorithm.

When the initial cluster centers have been determined with the use of the *buckshot* or *fractionation* algorithms, one can apply the k-means algorithm with the seeds obtained in the first step. Each document is assigned to the nearest of the k cluster centers. The centroid of each such cluster is determined as the concatenation of the documents in that cluster. Furthermore, the less frequent words of each centroid are removed. These centroids replace the seeds from the previous iteration. This process can be iteratively repeated to refine the cluster centers. Typically, only a small constant number of iterations is required because the greatest improvements occur only in the first few iterations. This ensures that the overall running time of each of the first and second phases is $O(k \cdot n)$.

It is also possible to use a number of enhancements after the second clustering phase. These enhancements are as follows:

- *Split operation:* The process of splitting can be used to further refine the clusters into groups of better granularity. This can be achieved by applying the buckshot procedure on the individual documents in a cluster by using $k = 2$ and then reclustering around these centers. This entire procedure requires $O(k \cdot n_i)$ time for a cluster containing n_i documents, and therefore splitting all the groups requires $O(k \cdot n)$ time. However, it is not necessary to split *all* the groups. Instead, only a subset of the groups can be split. These are the groups that are not very coherent and contain documents of a disparate nature. To measure the coherence of a group, the self-similarity of the documents in the cluster is computed. This self-similarity provides an understanding of the underlying coherence. This quantity can be computed either in terms of the average similarity of the documents in a cluster to its centroid or in terms of the average similarity of the cluster documents to each other. The split criterion can then be applied selectively only to those clusters that have low self-similarity. This helps in creating more coherent clusters.

- *Join operation:* The join operation merges similar clusters into a single cluster. To perform the merging, the *topical* words of each cluster are computed, as the most frequent words in the centroid of the cluster. Two clusters are considered similar if there is significant overlap between the topical words of the two clusters.

The *scatter/gather* approach is effective because of its ability to combine hierarchical and k-means algorithms.

13.3.2 Probabilistic Algorithms

Probabilistic text clustering can be considered an unsupervised version of the naive Bayes classification method discussed in Sect. 10.5.1 of Chap. 10. It is assumed that the documents need to be assigned to one of k clusters $\mathcal{G}_1 \ldots \mathcal{G}_k$. The basic idea is to use the following generative process:

1. Select a cluster $\mathcal{G}_m$, where $m \in \{1 \ldots k\}$.
2. Generate the term distribution of $\mathcal{G}_m$ based on a generative model. Examples of such models for text include the Bernoulli model or the multinomial model.

The observed data are then used to estimate the parameters of the Bernoulli or multinomial distributions in the generative process. This section will discuss the Bernoulli model.

The clustering is done in an iterative way with the EM algorithm, where cluster assignments of documents are determined from conditional word distributions in the E-step with the Bayes rule, and the conditional word distributions are inferred from cluster assignments in the M-step. For initialization, the documents are randomly assigned to clusters. The initial prior probabilities $P(\mathcal{G}_m)$ and conditional feature distributions $P(w_j|\mathcal{G}_m)$ are estimated from the statistical distribution of this random assignment. A Bayes classifier is used to estimate the posterior probability $P(\mathcal{G}_m|\overline{X})$ in the E-step. The Bayes classifier commonly uses either a Bernoulli model or the multinomial model discussed later in this chapter. The posterior probability $P(\mathcal{G}_m|\overline{X})$ of the Bayes classifier can be viewed as a soft assignment probability of document $\overline{X}$ to the mth mixture component $\mathcal{G}_m$. The conditional feature distribution $P(w_j|\mathcal{G}_m)$ for word w_j is estimated from these posterior probabilities in the M-step as follows:

$$P(w_j|\mathcal{G}_m) = \frac{\sum_{\overline{X}} P(\mathcal{G}_m|\overline{X}) \cdot I(\overline{X}, w_j)}{\sum_{\overline{X}} P(\mathcal{G}_m|\overline{X})} \tag{13.5}$$

Here, $I(\overline{X}, w_j)$ is an indicator variable that takes on the value of 1, if the word w_j is present in $\overline{X}$, and 0, otherwise. As in the Bayes classification method, the same Laplacian smoothing approach may be incorporated to reduce overfitting. The prior probabilities $P(\mathcal{G}_m)$ for each cluster may also be estimated by computing the average assignment probability of all documents to $\mathcal{G}_m$. This completes the description of the M-step of the EM algorithm. The next E-step uses these modified values of $P(w_j|\mathcal{G}_m)$ and the prior probability to derive the posterior Bayes probability with a standard Bayes classifier. Therefore, the following two iterative steps are repeated to convergence:

1. (E-step) Estimate posterior probability of membership of documents to clusters using Bayes rule:

$$P(\mathcal{G}_m|\overline{X}) \propto P(\mathcal{G}_m) \prod_{w_j \in \overline{X}} P(w_j|\mathcal{G}_m) \prod_{w_j \notin \overline{X}} (1 - P(w_j|\mathcal{G}_m)) . \tag{13.6}$$

   The aforementioned Bayes rule assumes a Bernoulli generative model. Note that Eq. 13.6 is identical to naive Bayes posterior probability estimation for classification. The multinomial model, which is discussed later in this chapter, may also be used. In such a case, the aforementioned posterior probability definition of Eq. 13.6 is replaced by the multinomial Bayes classifier.

2. (M-step) Estimate conditional distribution $P(w_j|\mathcal{G}_m)$ of words (Eq. 13.5) and prior probabilities $P(\mathcal{G}_m)$ of different clusters using the estimated probabilities in the E-step.

At the end of the process, the estimated value of $P(\mathcal{G}_m|\overline{X})$ provides a cluster assignment probability and the estimated value of $P(w_j|\mathcal{G}_m)$ provides the term distribution of each cluster. This can be viewed as a probabilistic variant of the notion of cluster digest discussed earlier. Therefore, the probabilistic method provides dual insights about cluster membership and the words relevant to each cluster.

13.3.3 Simultaneous Document and Word Cluster Discovery

The probabilistic algorithm discussed in the previous section can simultaneously discover document and word clusters. As discussed in Sect. 7.4 of Chap. 7 on high-dimensional clustering methods, this is important in the high-dimensional case because clusters are best characterized in terms of both rows and columns *simultaneously*. In the text domain, the additional advantage of such methods is that the topical words of a cluster provide *semantic* insights about that cluster. Another example is the nonnegative matrix factorization method, discussed in Sect. 6.8 of Chap. 6. This approach is very popular in the text domain because the factorized matrices have a natural interpretation for text data. This approach can simultaneously discover word clusters and document clusters that are represented by the columns of the two factorized matrices. This is also closely related to the concept of *co-clustering*.

13.3.3.1 Co-clustering

Co-clustering is most effective for nonnegative matrices in which many entries have zero values. In other words, the matrix is sparsely populated. This is the case for text data. Co-clustering methods can also be generalized to dense matrices, although these techniques are not relevant to the text domain. Co-clustering is also sometimes referred to as *bi-clustering* or *two-mode clustering* because of its exploitation of both "modes" (words and documents). While the co-clustering method is presented here in the context of text data, the broader approach is also used in the biological domain with some modifications.

The idea in co-clustering is to rearrange the rows and columns in the data matrix so that most of the nonzero entries become arranged into blocks. In the context of text data, this matrix is the $n \times d$ document term matrix D, where rows correspond to documents and columns correspond to words. Thus, the ith cluster is associated with a set of rows $\mathcal{R}_i$ (documents), and a set of columns $\mathcal{V}_i$ (words). The rows $\mathcal{R}_i$ are disjoint from one another over different values of i, and the columns $\mathcal{V}_i$ are also disjoint from one another over different values of i. Thus, the co-clustering method simultaneously leads to document clusters and word clusters. From an intuitive perspective, the words representing the columns of $\mathcal{V}_i$ are the most relevant (or topical) words for cluster $\mathcal{R}_i$. The set $\mathcal{V}_i$ therefore defines a cluster digest of $\mathcal{R}_i$.

In the context of text data, word clusters are just as important as document clusters because they provide insights about the topics of the underlying collection. Most of the methods discussed in this book for document clustering, such as the *scatter/gather* method, probabilistic methods, and nonnegative matrix factorization (see Sect. 6.8 of Chap. 6, produce word clusters (or cluster digests) in addition to document clusters. However, the words in the different clusters are *overlapping* in these algorithms, whereas document clusters are non overlapping in all algorithms except for the probabilistic (soft) EM method. In co-clustering, the word clusters and document clusters are *both* non overlapping. Each document and word is strictly associated with a particular cluster. One nice characteristic of co-clustering is that it explicitly explores the duality between word clusters and document clusters. Coherent word clusters can be shown to induce coherent document clusters and vice versa. For example, if meaningful word clusters were already available, then one might be able to cluster documents by assigning each document to the word cluster with which it has the most words in common. In co-clustering, the goal is to do this *simultaneously* so that word clusters and document clusters depend on each other in an optimal way.

	CHAMPION	ELECTRON	TROPHY	RELATIVITY	QUANTUM	TOURNAMENT
D_1	2	0	1	1	0	3
D_2	0	2	0	1	3	0
D_3	1	3	0	1	2	0
D_4	2	0	2	0	0	3
D_5	0	2	1	1	3	0
D_6	1	0	2	0	0	3

(a) Document-term matrix

SPORTS CO-CLUSTER

PHYSICS CO-CLUSTER

	CHAMPION	TROPHY	TOURNAMENT	ELECTRON	RELATIVITY	QUANTUM
D_1	2	1	3	0	1	0
D_4	2	2	3	0	0	0
D_6	1	2	3	0	0	0
D_2	0	0	0	2	1	3
D_3	1	0	0	3	1	2
D_5	0	1	0	2	1	3

(b) Re-arranged document-term matrix

Figure 13.1: Illustrating row and column reordering in co-clustering

To illustrate this point, a toy example[2] of a 6×6 document-word matrix has been illustrated in Fig. 13.1a. The entries in the matrix correspond to the word frequencies in six documents denoted by $D_1 \ldots D_6$. The six words in this case are *champion*, *electron*, *trophy*, *relativity*, *quantum*, and *tournament*. It is easy to see that some of the words are from sports-related topics, whereas other words are from science-related topics. Note that the nonzero entries in the matrix of Fig. 13.1a seem to be arranged randomly. It should be noted that the documents $\{D_1, D_4, D_6\}$ seem to contain words relating to sports topics, whereas the documents $\{D_2, D_3, D_5\}$ seem to contain words relating to scientific topics. However, this is not evident from the random distribution of the entries in Fig. 13.1a. On the other hand, if the rows and columns were permuted, so that all the sports-related rows/columns occur earlier than all the science-related rows/columns, then the resulting matrix is shown in Fig. 13.1b. In this case, there is a clear block structure to the entries, in which disjoint rectangular blocks contain most of the nonzero entries. These rectangular blocks are shaded in Fig. 13.1b. The goal is to minimize the weights of the nonzero entries outside these shaded blocks.

How, then, can the co-clustering problem be solved? The simplest solution is to convert the problem to a bipartite graph partitioning problem, so that the aggregate weight of the nonzero entries in the nonshaded regions is equal to the aggregate weight of the edges across the partitions. A node set N_d is created, in which each node represents a document in the collection. A node set N_w is created, in which each node represents a word in the collection. An undirected bipartite graph $G = (N_d \cup N_w, A)$ is created, such that an edge (i, j) in A corresponds to a nonzero entry in the matrix, where $i \in N_d$ and $j \in N_w$. The weight of an edge is equal to the frequency of the term in the document. The bipartite graph for the co-cluster of Fig. 13.1 is illustrated in Fig. 13.2. A partitioning of this graph represents a simultaneous partitioning of the rows and columns. In this case, a 2-way partitioning has been illustrated for simplicity, although a k-way partitioning could be constructed in general. Note that each partition contains a set of documents and a corresponding set of words. It is easy to see that the corresponding documents and words in each graph partition of Fig. 13.2 represent the shaded areas in Fig. 13.1b. It is also easy to see that the weight

[2]While the document-term matrix is square in this specific toy example, this might not be the case in general because the corpus size n, and the lexicon size d are generally different.

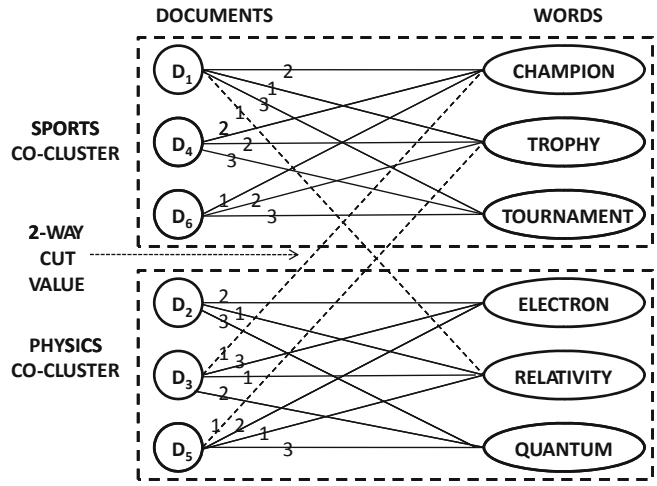


Figure 13.2: Graph partitioning for co-clustering

of edges across the partition represents the weight of the nonzero entries in Fig. 13.1b. Therefore, a k-way co-clustering problem can be converted to a k-way graph partitioning problem. The overall co-clustering approach may be described as follows:

1. Create a graph $G = (N_d \cup N_w, A)$ with nodes in N_d representing documents, nodes in N_w representing words, and edges in A with weights representing nonzero entries in matrix D.

2. Use a k-way graph partitioning algorithm to partition the nodes in $N_d \cup N_w$ into k groups.

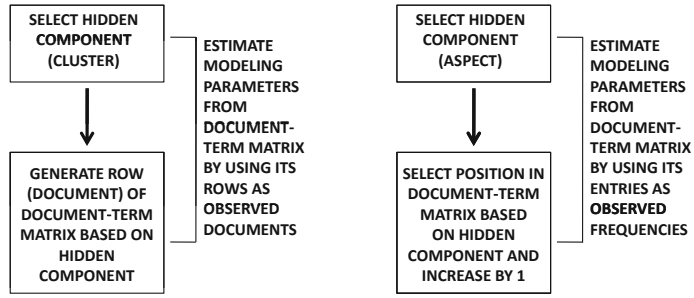
3. Report row–column pairs $(\mathcal{R}_i \mathcal{V}_i)$ for $i \in \{1 \ldots k\}$. Here, $\mathcal{R}_i$ represents the rows corresponding to nodes in N_d for the ith cluster, and $\mathcal{V}_i$ represents the columns corresponding to the nodes in N_w for the ith cluster.

It remains to be explained, how the k-way graph partitioning may be performed. The problem of graph partitioning is addressed in Sect. 19.3 of Chap. 19. Any of these algorithms may be leveraged to determine the required graph partitions. Specialized methods for bipartite graph partitioning are also discussed in the bibliographic notes.

13.4 Topic Modeling

Topic modeling can be viewed as a probabilistic version of the latent semantic analysis (*LSA*) method, and the most basic version of the approach is referred to as *Probabilistic Latent Semantic Analysis (PLSA)*. It provides an alternative method for performing dimensionality reduction and has several advantages over traditional *LSA*.

Probabilistic latent semantic analysis is an expectation maximization-based mixture modeling algorithm. However, the way in which the EM algorithm is used is different than the other examples of the EM algorithm in this book. This is because the underlying generative process is different, and is optimized to discovering the *correlation structure* of the words rather than the clustering structure of the documents. This is because the approach



(a) EM-clustering (section 13.3.2) (b) *PLSA*

Figure 13.3: Varying generative process of EM-clustering and *PLSA*

can be viewed as a probabilistic variant of *SVD* and *LSA*, rather than a probabilistic variant of clustering. Nevertheless, soft clusters can also be generated with the use of this method. There are many other dimensionality reduction methods, such as nonnegative matrix factorization, which are intimately related to clustering. *PLSA* is, in fact, a nonnegative matrix factorization method with a maximum-likelihood objective function.

In most of the EM clustering algorithms of this book, a mixture component (cluster) is selected, and then the data record is generated based on a particular form of the distribution of that component. An example is the Bernoulli clustering model, which is discussed in Sect. 13.3.2. In *PLSA*, the generative process[3] is inherently designed for dimensionality reduction rather than clustering, and different parts of the same document can be generated by different mixture components. It is assumed that there are k *aspects* (or latent topics) denoted by $\mathcal{G}_1 \ldots \mathcal{G}_k$. The generative process builds the document-term matrix as follows:

1. Select a latent component (aspect) $\mathcal{G}_m$ with probability $P(\mathcal{G}_m)$.
2. Generate the indices (i, j) of a document–word pair $(\overline{X_i}, w_j)$ with probabilities $P(\overline{X_i}|\mathcal{G}_m)$ and $P(w_j|\mathcal{G}_m)$, respectively. Increment the frequency of entry (i, j) in the document-term matrix by 1. The document and word indices are generated in a probabilistically independent way.

All the parameters of this generative process, such as $P(\mathcal{G}_m)$, $P(\overline{X_i}|\mathcal{G}_m)$, and $P(w_j|\mathcal{G}_m)$, need to be estimated from the observed frequencies in the $n \times d$ document-term matrix.

Although the aspects $\mathcal{G}_1 \ldots \mathcal{G}_k$ are analogous to the clusters of Sect. 13.3.2, they are not the same. Note that each iteration of the generative process of Sect. 13.3.2 creates the final frequency vector of an *entire row* of the document-term matrix. In *PLSA*, even a single matrix entry may have frequency contributions from various mixture components. Indeed, even in deterministic latent semantic analysis, a document is expressed as a linear combination of different latent directions. Therefore, the interpretation of each mixture component as a cluster is more direct in the method of Sect. 13.3.2. The generative differences between these models are illustrated in Fig. 13.3. Nevertheless, *PLSA* can also be used for clustering because of the highly interpretable and nonnegative nature of the underlying latent factorization. The relationship with and applicability to clustering will be discussed later.

[3]The original work [271] uses an asymmetric generative process, which is equivalent to the (simpler) symmetric generative process discussed here.

An important assumption in *PLSA* is that the selected documents and words are *conditionally* independent after the latent topical component $\mathcal{G}_m$ has been fixed. In other words, the following is assumed:

$$P(\overline{X_i}, w_j|\mathcal{G}_m) = P(\overline{X_i}|\mathcal{G}_m) \cdot P(w_j|\mathcal{G}_m) \tag{13.7}$$

This implies that the joint probability $P(\overline{X_i}, w_j)$ for selecting a document–word pair can be expressed in the following way:

$$P(\overline{X_i}, w_j) = \sum_{m=1}^{k} P(\mathcal{G}_m) \cdot P(\overline{X_i}, w_j|\mathcal{G}_m) = \sum_{m=1}^{k} P(\mathcal{G}_m) \cdot P(\overline{X_i}|\mathcal{G}_m) \cdot P(w_j|\mathcal{G}_m). \tag{13.8}$$

It is important to note that *local* independence between documents and words within a latent component does not imply *global* independence between the same pair over the entire corpus. The local independence assumption is useful in the derivation of EM algorithm.

In *PLSA*, the posterior probability $P(\mathcal{G}_m|\overline{X_i}, w_j)$ of the latent component associated with a particular *document–word pair* is estimated. The EM algorithm starts by initializing $P(\mathcal{G}_m)$, $P(\overline{X_i}|\mathcal{G}_m)$, and $P(w_j|\mathcal{G}_m)$ to $1/k$, $1/n$, and $1/d$, respectively. Here, k, n, and d denote the number of clusters, number of documents, and number of words, respectively. The algorithm iteratively executes the following E- and M-steps to convergence:

1. (E-step) Estimate posterior probability $P(\mathcal{G}_m|\overline{X_i}, w_j)$ in terms of $P(\mathcal{G}_m)$, $P(\overline{X_i}|\mathcal{G}_m)$, and $P(w_j|\mathcal{G}_m)$.

2. (M-step) Estimate $P(\mathcal{G}_m)$, $P(\overline{X_i}|\mathcal{G}_m)$ and $P(w_j|\mathcal{G}_m)$ in terms of the posterior probability $P(\mathcal{G}_m|\overline{X_i}, w_j)$, and observed data about word-document co-occurrence using log-likelihood maximization.

These steps are iteratively repeated to convergence. It now remains to discuss the details of the E-step and the M-step. First, the E-step is discussed. The posterior probability estimated in the E-step can be expanded using the Bayes rule:

$$P(\mathcal{G}_m|\overline{X_i}, w_j) = \frac{P(\mathcal{G}_m) \cdot P(\overline{X_i}, w_j|\mathcal{G}_m)}{P(\overline{X_i}, w_j)}. \tag{13.9}$$

The numerator of the right-hand side of the aforementioned equation can be expanded using Eq. 13.7, and the denominator can be expanded using Eq. 13.8:

$$P(\mathcal{G}_m|\overline{X_i}, w_j) = \frac{P(\mathcal{G}_m) \cdot P(\overline{X_i}|\mathcal{G}_m) \cdot P(w_j|\mathcal{G}_m)}{\sum_{r=1}^{k} P(\mathcal{G}_r) \cdot P(\overline{X_i}|\mathcal{G}_r) \cdot P(w_j|\mathcal{G}_r)}. \tag{13.10}$$

This shows that the E-step can be implemented in terms of the estimated values $P(\mathcal{G}_m)$, $P(\overline{X_i}|\mathcal{G}_m)$, and $P(w_j|\mathcal{G}_m)$.

It remains to show how these values can be estimated using the *observed* word-document co-occurrences in the M-step. The posterior probabilities $P(\mathcal{G}_m|\overline{X_i}, w_j)$ may be viewed as weights attached with word-document co-occurrence pairs for each aspect $\mathcal{G}_m$. These weights can be leveraged to estimate the values $P(\mathcal{G}_m)$, $P(\overline{X_i}|\mathcal{G}_m)$, and $P(w_j|\mathcal{G}_m)$ for each aspect using maximization of the log-likelihood function. The details of the log-likelihood function, and the differential calculus associated with the maximization process will not be discussed here. Rather, the final estimated values will be presented directly. Let $f(\overline{X_i}, w_j)$ represent

the observed frequency of the occurrence of word w_j in document $\overline{X_i}$ in the corpus. Then, the estimations in the M-step are as follows:

$$P(\overline{X_i}|\mathcal{G}_m) \propto \sum_{w_j} f(\overline{X_i}, w_j) \cdot P(\mathcal{G}_m|\overline{X_i}, w_j) \quad \forall i \in \{1 \ldots n\}, m \in \{1 \ldots k\} \tag{13.11}$$

$$P(w_j|\mathcal{G}_m) \propto \sum_{\overline{X_i}} f(\overline{X_i}, w_j) \cdot P(\mathcal{G}_m|\overline{X_i}, w_j) \quad \forall j \in \{1 \ldots d\}, m \in \{1 \ldots k\} \tag{13.12}$$

$$P(\mathcal{G}_m) \propto \sum_{\overline{X_i}} \sum_{w_j} f(\overline{X_i}, w_j) \cdot P(\mathcal{G}_m|\overline{X_i}, w_j) \quad \forall m \in \{1 \ldots k\}. \tag{13.13}$$

Each of these estimations may be scaled to a probability by ensuring that they sum to 1 over all the outcomes for that random variable. This scaling corresponds to the constant of proportionality associated with the "$\propto$" notation in the aforementioned equations. Furthermore, these estimations can be used to decompose the original document-term matrix into a product of three matrices, which is very similar to *SVD/LSA*. This relationship will be explored in the next section.

13.4.1 Use in Dimensionality Reduction and Comparison with Latent Semantic Analysis

The three key sets of parameters estimated in the M-step are $P(\overline{X_i}|\mathcal{G}_m)$, $P(w_j|\mathcal{G}_m)$, and $P(\mathcal{G}_m)$, respectively. These sets of parameters provide an *SVD*-like matrix factorization of the $n \times d$ document-term matrix D. Assume that the document-term matrix D is scaled by a constant to sum to an aggregate probability value of 1. Therefore, the (i, j)th entry of D can be viewed as an *observed instantiation* of the probabilistic quantity $P(\overline{X_i}, w_j)$. Let Q_k be the $n \times k$ matrix, for which the (i, m)th entry is $P(\overline{X_i}|\mathcal{G}_m)$, let Σ_k be the $k \times k$ diagonal matrix for which the mth diagonal entry is $P(\mathcal{G}_m)$, and let P_k be the $d \times k$ matrix for which the (j, m)th entry is $P(w_j|\mathcal{G}_m)$. Then, the (i, j)th entry $P(\overline{X_i}, w_j)$ of the matrix D can be expressed in terms of the entries of the aforementioned matrices according to Eq. 13.8, which is replicated here:

$$P(\overline{X_i}, w_j) = \sum_{m=1}^{k} P(\mathcal{G}_m) \cdot P(\overline{X_i}|\mathcal{G}_m) \cdot P(w_j|\mathcal{G}_m). \tag{13.14}$$

This LHS of the equation is equal to the (i, j)th entry of D, whereas the RHS of the equation is the (i, j)th entry of the matrix product $Q_k \Sigma_k P_k^T$. Depending on the number of components k, the LHS can only approximate the matrix D, which is denoted by D_k. By stacking up the $n \times d$ conditions of Eq. 13.14, the following matrix condition is obtained:

$$D_k = Q_k \Sigma_k P_k^T. \tag{13.15}$$

It is instructive to note that the matrix decomposition in Eq. 13.15 is similar to that in *SVD/LSA* (cf. Eq. 2.12 of Chap. 2). Therefore, as in *LSA*, D_k is an approximation of the document-term matrix D, and the transformed representation in k-dimensional space is given by $Q_k \Sigma_k$. However, the transformed representations will be different in *PLSA* and *LSA*. This is because different objective functions are optimized in the two cases. *LSA* minimizes the mean-squared error of the approximation, whereas *PLSA* maximizes the log-likelihood fit to a probabilistic generative model. One advantage of *PLSA* is that the entries of Q_k and P_k and the transformed coordinate values are nonnegative and have clear

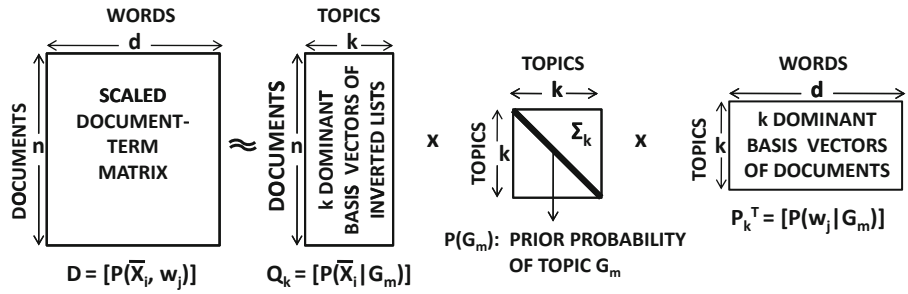


Figure 13.4: Matrix factorization of *PLSA*

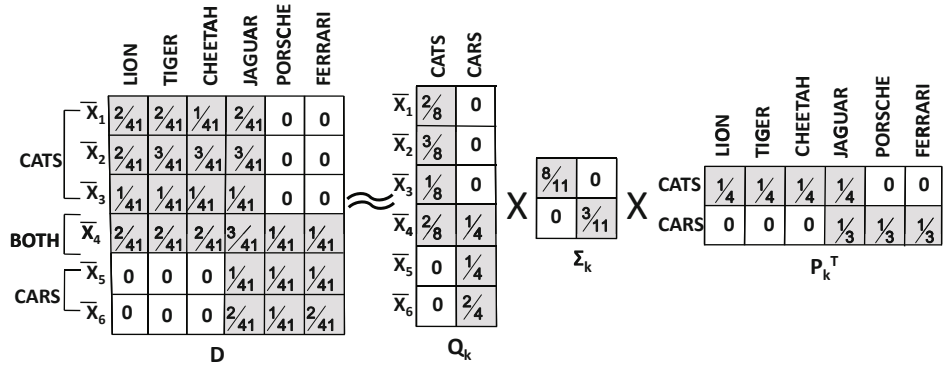


Figure 13.5: An example of *PLSA* (Revisiting Fig. 6.22 of Chap. 6)

probabilistic interpretability. By examining the probability values in each column of P_k, one can immediately infer the topical words of the corresponding aspect. This is not possible in *LSA*, where the entries in the corresponding matrix P_k do not have clear probabilistic significance and may even be negative. One advantage of *LSA* is that the transformation can be interpreted in terms of the rotation of an orthonormal axis system. In *LSA*, the columns of P_k are a set of orthonormal vectors representing this rotated basis. This is not the case in *PLSA*. Orthogonality of the basis system in *LSA* enables straightforward projection of *out-of-sample* documents (i.e., documents not included in D) onto the new rotated axis system.

Interestingly, as in *SVD/LSA*, the latent properties of the transpose of the document matrix are revealed by *PLSA*. Each row of $P_k\Sigma_k$ can be viewed as the transformed coordinates of the *vertical or inverted list representation* (rows of the transpose) of the document matrix D in the basis space defined by columns of Q_k. These complementary properties are illustrated in Fig. 13.4. *PLSA* can also be viewed as a kind of nonnegative matrix factorization method (cf. Sect. 6.8 of Chap. 6) in which matrix elements are interpreted as probabilities and the maximum-likelihood estimate of a generative model is maximized rather than minimizing the Frobenius norm of the error matrix.

An example of an approximately optimal *PLSA* matrix factorization of a toy 6×6 example, with 6 documents and 6 words, is illustrated in Fig. 13.5. This example is the same (see Fig. 6.22) as the one used for nonnegative matrix factorization (*NMF*) in Chap. 6. Note that the factorizations in the two cases are very similar except that all basis vectors

are normalized to sum to 1 in *PLSA*, and the dominance of the basis vectors is reflected in a separate diagonal matrix containing the prior probabilities. Although the factorization presented here for *PLSA* is identical to that of *NMF* for intuitive understanding, the factorizations will usually be slightly different[4] because of the difference in objective functions in the two cases. Also, most of the entries in the factorized matrices will not be exactly 0 in a real example, but many of them might be quite small.

As in *LSA*, the problems of synonymy and polysemy are addressed by *PLSA*. For example, if an aspect $\mathcal{G}_1$ explains the topic of cats, then two documents $\overline{X}$ and $\overline{Y}$ containing the words "cat" and "kitten," respectively, will have positive values of the transformed coordinate for aspect $\mathcal{G}_1$. Therefore, similarity computations between these documents will be improved in the transformed space. A word with multiple meanings (polysemous word) may have positive components in different aspects. For example, a word such as "jaguar" can either be a cat or a car. If $\mathcal{G}_1$ be an aspect that explains the topic of cats, and $\mathcal{G}_2$ is an aspect that explains the topic of cars, then both $P(\text{"jaguar"}|\mathcal{G}_1)$ and $P(\text{"jaguar"}|\mathcal{G}_2)$ may be highly positive. However, the other words in the document will provide the context necessary to reinforce one of these two aspects. A document $\overline{X}$ that is mostly about cats will have a high value of $P(\overline{X}|\mathcal{G}_1)$, whereas a document $\overline{Y}$ that is mostly about cars will have a high value of $P(\overline{Y}|\mathcal{G}_2)$. This will be reflected in the matrix $Q_k = [P(\overline{X_i}|\mathcal{G}_m)]_{n \times k}$ and the new transformed coordinate representation $Q_k \Sigma_k$. Therefore, the computations will also be robust in terms of adjusting for polysemy effects. In general, semantic concepts will be amplified in the transformed representation $Q_k \Sigma_k$. Therefore, many data mining applications will perform more robustly in terms of the $n \times k$ transformed representation $Q_k \Sigma_k$ rather than the original $n \times d$ document-term matrix.

13.4.2 Use in Clustering and Comparison with Probabilistic Clustering

The estimated parameters have intuitive interpretations in terms of clustering. In the Bayes model for clustering (Fig. 13.3a), the generative process is optimized to clustering documents, whereas the generative process in topic modeling (Fig. 13.3b) is optimized to discovering the latent semantic components. The latter can be shown to cluster document–word *pairs*, which is different from clustering documents. Therefore, although the same parameter set $P(w_j|\mathcal{G}_m)$ and $P(\overline{X}|\mathcal{G}_m)$ is estimated in the two cases, qualitatively different results will be obtained. The model of Fig. 13.3a generates a document from a unique hidden component (cluster), and the final soft clustering is a result of uncertainty in *estimation* from observed data. On the other hand, in the probabilistic latent semantic model, different parts of the same document may be generated by different aspects, even at the generative *modeling* level. Thus, documents are not generated by individual mixture components, but by a combination of mixture components. In this sense, *PLSA* provides a more realistic model because the diverse words of an unusual document discussing both cats and cars (see Fig. 13.5) can be generated by distinct aspects. In Bayes clustering, even though such a document is generated *in entirety* by one of the mixture components, it may have similar assignment (posterior) probabilities with respect to two or more clusters because of *estimation uncertainty*. This difference is because *PLSA* was originally intended as a data transformation and dimensionality reduction method, rather than as a clustering method. Nevertheless, good document clusters can usually be derived from *PLSA* as well. The value $P(\mathcal{G}_m|\overline{X_i})$ provides an assignment probability of the document $\overline{X_i}$ to aspect (or "cluster")

[4]The presented factorization for *PLSA* is approximately optimal, but not exactly optimal.

$\mathcal{G}_m$ and can be derived from the parameters estimated in the M-step using the Bayes rule as follows:

$$P(\mathcal{G}_m|\overline{X_i}) = \frac{P(\mathcal{G}_m) \cdot P(\overline{X_i}|\mathcal{G}_m)}{\sum_{r=1}^{k} P(\mathcal{G}_r) \cdot P(\overline{X_i}|\mathcal{G}_r)}. \tag{13.16}$$

Thus, the *PLSA* approach can also be viewed a *soft* clustering method that provides assignment probabilities of documents to clusters. In addition, the quantity $P(w_j|\mathcal{G}_m)$, which is estimated in the M-step, provides probabilistic information about the probabilistic affinity of different words to aspects (or *topics*). The terms with the highest probability values for a specific aspect $\mathcal{G}_m$ can be viewed as a *cluster digest* for that topic.

As the *PLSA* approach also provides a multidimensional $n \times k$ coordinate representation $Q_k\Sigma_k$ of the documents, a different way of performing the clustering would be to represent the documents in this new space and use a k-means algorithm on the transformed corpus. Because the noise impact of synonymy and polysemy has been removed by *PLSA*, the k-means approach will generally be more effective on the reduced representation than on the original corpus.

13.4.3 Limitations of PLSA

Although the *PLSA* method is an intuitively sound model for probabilistic modeling, it does have a number of practical drawbacks. The number of parameters grows linearly with the number of documents. Therefore, such an approach can be slow and may overfit the training data because of the large number of estimated parameters. Furthermore, while *PLSA* provides a generative model of document–word pairs in the training data, it cannot easily assign probabilities to previously unseen documents. Most of the other EM mixture models discussed in this book, such as the probabilistic Bayes model, are much better at assigning probabilities to previously unseen documents. To address these issues, *Latent Dirichlet Allocation (LDA)* was defined. This model uses Dirichlet priors on the topics, and generalizes relatively easily to new documents. In this sense, *LDA* is a fully generative model. The bibliographic notes contain pointers to this model.

13.5 Specialized Classification Methods for Text

As in clustering, classification algorithms are affected by the nonnegative, sparse and high-dimensional nature of text data. An important effect of sparsity is that the presence of a word in a document is more informative than the absence of the word. This observation has implications for classification methods such as the Bernoulli model used for Bayes classification that treat the presence and absence of a word in a symmetric way.

Popular techniques in the text domain include instance-based methods, the Bayes classifier, and the SVM classifier. The Bayes classifier is very popular because Web text is often combined with other types of features such as URLs or side information. It is relatively easy to incorporate these features into the Bayes classifier. The sparse high-dimensional nature of text also necessitates the design of more refined multinomial Bayes models for the text domain. SVM classifiers are also extremely popular for text data because of their high accuracy. The major issue with the use of the SVM classifier is that the high-dimensional nature of text necessitates performance enhancements to such classifiers. In the following, some of these algorithms will be discussed.

13.5.1 Instance-Based Classifiers

Instance-based classifiers work surprisingly well for text, especially when a preprocessing phase of clustering or dimensionality reduction is performed. The simplest form of the nearest neighbor classifier returns the dominant class label of the top-k nearest neighbors with the cosine similarity. Weighting the vote with the cosine similarity value often provides more robust results. However because of the sparse and high-dimensional nature of text collections, this basic procedure can be modified in two ways to improve both the efficiency and the effectiveness. The first method uses dimensionality reduction in the form of latent semantic indexing. The second method uses fine-grained clustering to perform centroid-based classification.

13.5.1.1 Leveraging Latent Semantic Analysis

A major source of error in instance-based classification is the noise inherent in text collections. This noise is often a result of *synonymy* and *polysemy*. For example, the words *comical* and *hilarious* mean approximately the same thing. Polysemy refers to the fact that the same word may mean two different things. For example, the word *jaguar* could refer to a car or a cat. Typically, the significance of a word can be understood only in the context of other words in the document. These characteristics of text create challenges for classification algorithms because the computation of similarity with the use of word frequencies may not be completely accurate. For example, two documents containing the words *comical* and *hilarious*, respectively, may not be deemed sufficiently similar because of synonymy effects. In latent semantic indexing, dimensionality reduction is applied to the collection to reduce these effects.

Latent semantic analysis (LSA) is an approach that relies on singular value decomposition (SVD) to create a reduced representation for the text collection. The reader is advised to refer to Sect. 2.4.3.3 of Chap. 2 for details of SVD and LSA. The latent semantic analysis (LSA) method is an application of the SVD method to the $n \times d$ document-term matrix D, where d is the size of the lexicon, and n is the number of documents. The eigenvectors with the largest eigenvalues of the square $d \times d$ matrix $D^T D$ are used for data representation. The sparsity of the data set results in a low intrinsic dimensionality. Therefore, in the text domain, the reduction in dimensionality resulting from LSA is rather drastic. For example, it is not uncommon to be able to represent a corpus drawn on a lexicon of size 100,000 in less than 300 dimensions. The removal of the dimensions with small eigenvalues typically leads to a reduction in the noise effects of synonymy and polysemy. This data representation is no longer sparse and resembles multidimensional numeric data. A conventional k-nearest neighbor classifier with cosine similarity can be used on this transformed corpus. The LSA method does require an additional effort up front to create the eigenvectors.

13.5.1.2 Centroid-Based Classification

Centroid-based classification is a fast alternative to k-nearest neighbor classifiers. The basic idea is to use an off-the-shelf clustering algorithm to partition the documents of each class into clusters. The number of clusters derived from the documents of each class is proportional to the number of documents in that class. This ensures that the clusters in each class are of approximately the same granularity. Class labels are associated with individual clusters rather than the actual documents.

The cluster digests from the centroids are extracted by retaining only the most frequent words in that centroid. Typically, about 200 to 400 words are retained in each centroid. The

lexicon in each of these centroids provides a stable and topical representation of the subjects in each class. An example of the (weighted) word vectors for two classes corresponding to the labels "*Business schools*" and "*Law schools*" could be as follows:

1. *Business schools:* business (35), management (31), school (22), university (11), campus (15), presentation (12), student (17), market (11), ...
2. *Law schools:* law (22), university (11), school (13), examination (15), justice (17), campus (10), courts (15), prosecutor (22), student (15), ...

Typically, most of the noisy words have been truncated from the cluster digest. Similar words are represented in the same centroid, and words with multiple meanings can be represented in contextually different centroids. Therefore, this approach also indirectly addresses the issues of synonymy and polysemy, with the *additional* advantage that the k-nearest neighbor classification can be performed more efficiently with a smaller number of centroids. The dominant label from the top-k matching centroids, based on cosine similarity, is reported. Such an approach can provide comparable or better accuracy than the vanilla k-nearest neighbor classifier in many cases.

13.5.1.3 Rocchio Classification

The Rocchio method can be viewed as a special case of the aforementioned description of the centroid-based classifier. In this case, all documents belonging to the same class are aggregated into a *single* centroid. For a given document, the class label of the closest centroid is reported. This approach is obviously extremely fast because it requires a small constant number of similarity computations that is dependent on the number of classes in the data. On the other hand, the drawback is that the accuracy depends on the assumption of *class contiguity*. The class-contiguity assumption, as stated in [377], is as follows:

"*Documents in the same class form a contiguous region, and regions of different classes do not overlap.*"

Thus, Rocchio's method would not work very well if documents of the same class were separated into distinct clusters. In such cases, the centroid of a class of documents may not even lie in one of the clusters of that class. A bad case for Rocchio's method is illustrated in Fig. 13.6, in which two classes and four clusters are depicted. Each class is associated with two distinct clusters. In this case, the centroids for each of the classes are approximately the same. Therefore, the Rocchio method would have difficulty in distinguishing between the classes. On the other hand, a k-nearest neighbor classifier for small values of k, or a centroid-based classifier would perform quite well in this case. As discussed in Chap. 11, an increase in the value of k for a k-nearest neighbor classifier increases its bias. The Rocchio classifier can be viewed as a k-nearest neighbor classifier with a high value of k.

13.5.2 Bayes Classifiers

The Bayes classifier is described in Sect. 10.5.1 of Chap. 10. The particular classifier described was a binary (or *Bernoulli*) model in which the posterior probability of a document belonging to a particular class was computed using only the presence or the absence of a word. This special case corresponds to the fact that each feature (word) takes on the value of either 0 or 1 depending on whether or not it is present in the document. However, such an approach does not account for the frequencies of the words in the documents.

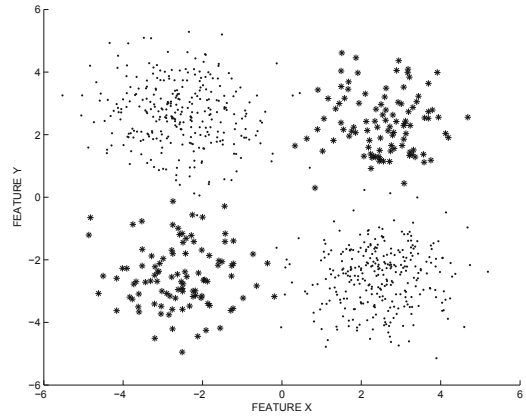


Figure 13.6: A bad case for the Rocchio method

13.5.2.1 Multinomial Bayes Model

A more general approach is to use a multinomial Bayes model, in which the frequencies of the words are used explicitly. The Bernoulli model is helpful mostly for cases where the documents are short, and drawn over a lexicon of small size. In the general case of documents of longer sizes over a large lexicon, the multinomial model is more effective. Before discussing the multinomial model, the Bernoulli model (cf. Sect. 10.5.1 of Chap. 10) will be revisited in the context of text classification.

Let C be the random variable representing the class variable of an unseen test instance, with d-dimensional feature values $\overline{X} = (a_1 \ldots a_d)$. For the Bernoulli model on text data, each value of a_i is 1 or 0, depending on whether or not the ith word of the lexicon is present in the document $\overline{X}$. The goal is to estimate the posterior probability $P(C = c|\overline{X} = (a_1 \ldots a_d))$. Let the random variables for the individual dimensions of $\overline{X}$ be denoted by $\overline{X} = (x_1 \ldots x_d)$. Then, it is desired to estimate the conditional probability $P(C = c|x_1 = a_1, \ldots x_d = a_d)$. Then, by using Bayes' theorem, the following equivalence can be inferred.

$$P(C = c|x_1 = a_1, \ldots x_d = a_d) = \frac{P(C = c)P(x_1 = a_1, \ldots x_d = a_d|C = c)}{P(x_1 = a_1, \ldots x_d = a_d)} \tag{13.17}$$

$$\propto P(C = c)P(x_1 = a_1, \ldots x_d = a_d|C = c) \tag{13.18}$$

$$\approx P(C = c) \prod_{i=1}^{d} P(x_i = a_i|C = c). \tag{13.19}$$

The last of the aforementioned relationships is based on the *naive* assumption of conditional independence. In the binary model discussed in Chap. 10, each attribute value a_i takes on the value of 1 or 0 depending on the presence or the absence of a word. Thus, if the fraction of the documents in class c containing word i is denoted by $p(i, c)$, then the value of $P(x_i = a_i|C = c)$ is estimated[5] as either $p(i, c)$ or $1 - p(i, c)$ depending upon whether a_i is 1 or 0, respectively. Note that this approach explicitly penalizes nonoccurrence of words in documents. Larger lexicon sizes will result in many words that are absent in a document. Therefore, the Bernoulli model may be dominated by word absence rather than

[5]The exact value will be slightly different because of Laplacian smoothing. Readers are advised to refer to Sect. 10.5.1 of Chap. 10.

word presence. Word absence is usually weakly related to class labels. This leads to greater noise in the evaluation. Furthermore, differential frequencies of words are ignored by this approach. Longer documents are more likely to have repeated words. The multinomial model is designed to address these issues.

In the multinomial model, the L terms in a document are treated as samples from a multinomial distribution. The total number of terms in the document (or document length) is denoted by $L = \sum_{j=1}^{d} a_i$. In this case, the value of a_i is assumed to be the raw frequency of the term in the document. The posterior class probabilities of a test document with the frequency vector $(a_1 \ldots a_d)$ are defined and estimated using the following generative approach:

1. Sample a class c with a class-specific prior probability.
2. Sample L terms *with replacement* from the term distribution of the chosen class c. The term distribution is defined using a multinomial model. The sampling process generates the frequency vector $(a_1 \ldots a_d)$. All training and test documents are assumed to be observed samples of this generative process. Therefore, all model parameters of the generative process are estimated from the training data.
3. *Test instance classification:* What is the posterior probability that the class c is selected in the first generative step, conditional on the *observed* word frequency $(a_1 \ldots a_d)$ in the test document?

When the sequential ordering of the L different samples are considered, the number of possible ways to sample the different terms to result in the representation $(a_1 \ldots a_d)$ is given by $\frac{L!}{\prod_{i:a_i>0} a_i!}$. The probability of *each* of these sequences is given by $\prod_{i:a_i>0} p(i,c)^{a_i}$, by using the naive independence assumption. In this case, $p(i,c)$ is estimated as the fractional number of occurrences of word i in class c *including repetitions*. Therefore, unlike the Bernoulli model, repeated presence of word i in a document belonging to class c will increase $p(i,c)$. If $n(i,c)$ is the number of occurrences of word i in all documents belonging to class c, then $p(i,c) = \frac{n(i,c)}{\sum_i n(i,c)}$. Then, the class conditional feature distribution is estimated as follows:

$$P(x_1 = a_1, \ldots x_d = a_d | C = c) \approx \frac{L!}{\prod_{i:a_i>0} a_i!} \prod_{i:a_i>0} p(i,c)^{a_i}. \tag{13.20}$$

Using the Bayes rule, the multinomial Bayes model computes the posterior probability for a test document as follows:

$$P(C = c | x_1 = a_1, \ldots x_d = a_d) \propto P(C = c) \cdot P(x_1 = a_1, \ldots x_d = a_d | C = c) \tag{13.21}$$

$$\approx P(C = c) \cdot \frac{L!}{\prod_{i:a_i>0} a_i!} \prod_{i:a_i>0} p(i,c)^{a_i} \tag{13.22}$$

$$\propto P(C = c) \cdot \prod_{i:a_i>0} p(i,c)^{a_i}. \tag{13.23}$$

The constant factor $\frac{L!}{\prod_{i:a_i>0} a_i!}$ has been removed from the last condition because it is the same across all classes. Note that in this case, the product on the right-hand side only uses those words i, for which a_i is strictly larger than 0. Therefore, nonoccurrence of words is ignored. In this case, we have assumed that each a_i is the raw frequency of a word, which is an integer. It is also possible to use the multinomial Bayes model with the tf-idf frequency of a word, in which the frequency a_i might be fractional. However, the generative explanation becomes less intuitive in such a case.

13.5.3 SVM Classifiers for High-Dimensional and Sparse Data

The number of terms in the Lagrangian dual of the SVM formulation scales with the square of the number of dimensions. The reader is advised to refer to Sect. 10.6 of Chap. 10, and 11.4.2 of Chap. 11 for the relevant discussions. While the *SVMLight* method in Sect. 11.4.2 of Chap. 11 addresses this issue by making changes to the algorithmic approach, it does not make modifications to the SVM formulation itself. Most importantly, the approach does not make any modifications to address the high dimensional and sparse nature of text data.

The text domain is high dimensional and sparse. Only a small subset of the dimensions take on nonzero values for a given text document. Furthermore, linear classifiers tend to work rather well for the text domain, and it is often not necessary to use the kernelized version of the classifier. Therefore, it is natural to focus on linear classifiers, and ask whether it is possible to improve the complexity of SVM classification further by using the special domain-specific characteristics of text. *SVMPerf* is a linear-time algorithm designed for text classification. Its training complexity is $O(n \cdot s)$, where s is the average number of nonzero attributes per training document in the collection.

To explain the approach, we first briefly recap the soft penalty-based SVM formulation introduced in Sect. 10.6 of Chap. 10. The problem definition, referred to as the optimization formulation (OP1), is as follows:

$$\text{(OP1): Minimize } \frac{||\overline{W}||^2}{2} + C\frac{\sum_{i=1}^n \xi_i}{n}$$
$$\text{subject to:}$$
$$y_i \overline{W} \cdot \overline{X_i} \geq 1 - \xi_i \quad \forall i$$
$$\xi_i \geq 0 \quad \forall i.$$

One difference from the conventional SVM formulation of Chap. 10 is that the constant term b is missing. The conventional SVM formulation uses the constraint $y_i(\overline{W} \cdot \overline{X_i} + b) \geq 1 - \xi_i$. The two formulations are, however, equivalent because it can be shown that adding a dummy feature with a constant value of 1 to each training instance has the same effect. The coefficient in $\overline{W}$ of this feature will be equal to b. Another minor difference from the conventional formulation is that the slack component in the objective function is scaled by a factor of n. This is not a significant difference either because the constant C can be adjusted accordingly. These minor variations in the notation are performed without loss of generality for algebraic simplicity.

The *SVMPerf* method reformulates this problem with a *single* slack variable ξ, and 2^n constraints that are generated by summing a random subset of the n constraints in (OP1). Let $\overline{U} = (u_1 \ldots u_n) \in \{0, 1\}^n$ represent the indicator vector for the constraints that are summed up to create this new synthetic constraint. An alternative formulation of the SVM model is as follows:

$$\text{(OP2): Minimize } \frac{||\overline{W}||^2}{2} + C\xi$$
$$\text{subject to:}$$
$$\frac{1}{n}\sum_{i=1}^n u_i y_i \overline{W} \cdot \overline{X_i} \geq \frac{\sum_{i=1}^n u_i}{n} - \xi \quad \forall\, \overline{U} \in \{0, 1\}^n$$
$$\xi \geq 0.$$

The optimization formulation (OP2) is different from (OP1) in that it has only one slack variable ξ but 2^n constraints that represent the sum of every subset of constraints in (OP1). It can be shown that a one-to-one correspondence exists between the solutions of (OP1) and (OP2).

Lemma 13.5.1 *A one-to-one correspondence exists between solutions of (OP1) and (OP2), with equal values of* $\overline{W} = \overline{W^*}$ *in both models, and* $\xi^* = \frac{\sum_{i=1}^n \xi_i^*}{n}$.

Proof: We will show that if the same value of $\overline{W}$ is fixed for (OP1), and (OP2), then it will lead to the same objective function value. The first step is to derive the slack variables in terms of this value of $\overline{W}$ for (OP1) and (OP2). For problem (OP1), it can be derived from the slack constraints that the optimal value of ξ_i is achieved for $\xi_i = \max\{0, 1 - y_i \overline{W} \cdot \overline{X_i}\}$ in order to minimize the slack penalty. For the problem OP2, a similar result for ξ can be obtained:

$$\xi = \max_{u_1 \ldots u_n} \left\{ \frac{\sum_{i=1}^n u_i}{n} - \frac{1}{n} \sum_{i=1}^n u_i y_i \overline{W} \cdot \overline{X_i} \right\}. \tag{13.24}$$

Because this function is linearly separable in u_i, one can push the maximum inside the summation, and independently optimize for each u_i:

$$\xi = \sum_{i=1}^n \max_{u_i} u_i \left\{ \frac{1}{n} - \frac{1}{n} y_i \overline{W} \cdot \overline{X_i} \right\}. \tag{13.25}$$

For optimality, the value of u_i should be picked as 1 for only the positive values of $\left\{\frac{1}{n} - \frac{1}{n} y_i \overline{W} \cdot \overline{X_i}\right\}$ and 0, otherwise. Therefore, one can show the following:

$$\xi = \sum_{i=1}^n \max \left\{ 0, \frac{1}{n} - \frac{1}{n} y_i \overline{W} \cdot \overline{X_i} \right\} \tag{13.26}$$

$$= \frac{1}{n} \sum_{i=1}^n \max \left\{ 0, 1 - y_i \overline{W} \cdot \overline{X_i} \right\} = \frac{\sum_{i=1}^n \xi_i}{n}. \tag{13.27}$$

This one-to-one correspondence between optimal values of $\overline{W}$ in (OP1) and (OP2) implies that the two optimization problems are equivalent. ■

Thus, by determining the optimal solution to problem (OP2), it is possible to determine the optimal solution to (OP1) as well. Of course, it is not yet clear, why (OP2) is a better formulation than (OP1). After all, problem (OP2) contains an exponential number of constraints, and it seems to be intractable to even *enumerate* the constraints, let alone solve them.

Even so, the optimization formulation (OP2) does have some advantages over (OP1). First, a single slack variable measures the feasibility of all the constraints. This implies that all constraints can be expressed in terms of $(\overline{W}, \xi)$. Therefore, if one were to solve the optimization problem with only a subset of the 2^n constraints and the remaining were satisfied to a precision of ϵ by $(\overline{W}, \xi)$, then it is guaranteed that $(\overline{W}, \xi + \epsilon)$ is feasible for the *full* set of constraints.

The key is to never use all the constraints *explicitly*. Rather, a small subset $\mathcal{WS}$ of the 2^n constraints is used as the working set. We start with an empty working set $\mathcal{WS}$. The corresponding optimization problem is solved, and the most violated constraint among the constraints not in $\mathcal{WS}$ is added to the working set. The vector $\overline{U}$ for the most violated constraint is relatively easy to find. This is done by setting u_i to 1, if $y_i \overline{W} \cdot \overline{X_i} < 1$, and 0 otherwise. Therefore, the iterative steps for adding to the working set $\mathcal{WS}$ are as follows:

1. Determine optimal solution $(\overline{W}, \xi)$ for objective function of (OP2) using only constraints in the working set $\mathcal{WS}$.
2. Determine most violated constraint among the 2^n constraints of (OP2) by setting u_1 to 1 if $y_i \overline{W} \cdot \overline{X_i} < 1$, and 0 otherwise.
3. Add the most violated constraint to $\mathcal{WS}$.

The termination criterion is the case when the most violated constraint is violated by no more than ϵ. This provides an approximate solution to the problem, depending on the desired precision level ϵ.

This algorithm has several desirable properties. It can be shown that the time required to solve the problem for a constant size working set $\mathcal{WS}$ is $O(n \cdot s)$, where n is the number of training examples, and s is the number of *nonzero attributes* per example. This is important for the text domain, where the number of non-zero attributes is small. Furthermore, the algorithm usually terminates in a small constant number of iterations. Therefore, the working set $\mathcal{WS}$ never exceeds a constant size, and the entire algorithm terminates in $O(n \cdot s)$ time.

13.6 Novelty and First Story Detection

The problem of first story detection is a popular one in the context of temporal text stream mining applications. The goal is to determine novelties from the underlying text stream based on the history of previous text documents in the stream. This problem is particularly important in the context of streams of news documents, where a first story on a new topic needs to be reported as soon as possible.

A simple approach is to compute the maximum similarity of the current document with all previous documents, and report the documents with very low maximum similarity values as novelties. Alternatively, the inverse of the maximum similarity value could be continuously reported as a streaming novelty score or alarm level. The major problem with this approach is that the stream size continuously increases with time, and one has to compute similarity with all previous documents. One possibility is to use reservoir sampling to maintain a constant sample of documents. The inverse of the maximum similarity of the document to any incoming document is reported as the novelty score. The major drawback of this approach is that similarity between individual pairs of documents is often not a stable representation of the aggregate trends. Text documents are sparse, and pairwise similarity often does not capture the impact of synonymy and polysemy.

13.6.1 Micro-clustering Method

The micro-clustering method can be used to maintain online clusters of the text documents. The idea is that micro-clustering simultaneously determines the clusters and novelties from the underlying text stream. The basic micro-clustering method is described in Sect. 12.4 of Chap. 12. The approach maintains k different cluster centroids, or cluster digests. For an incoming document, its similarity to all the centroids is computed. If this similarity is larger than a user-defined threshold, then the document is added to the cluster. The frequencies of the words in the corresponding centroid are updated, by adding the frequency of the word in the document to it. For each document, only the r most frequent words in the centroid are retained. The typical value of r varies between 200 and 400. On the other hand, when the incoming document is not sufficiently similar to one of the centroids, then it is reported

as a novelty, or as a first story. A new cluster is created containing the singleton document. To make room for the new centroid, one of the old centroids needs to be removed. This is achieved by maintaining the last update time of each cluster. The most stale cluster is removed. This algorithm provides the online ability to report the novelties in the text stream. The bibliographic notes contain pointers to more detailed versions of this method.

13.7 Summary

The text domain is sometimes challenging for mining purposes because of its sparse and high-dimensional nature. Therefore, specialized algorithms need to be designed.

The first step is the construction of a bag-of-words representation for text data. Several preprocessing steps need to be applied, such as stop-word removal, stemming, and the removal of digits from the representation. For Web documents, preprocessing techniques are also required to remove the anchor text and to extract text from the main block of the page.

Algorithms for problems such as clustering and classification need to be modified as well. For example, density-based methods are rarely used for clustering text. The k-means methods, hierarchical methods, and probabilistic methods can be suitably modified to work for text data. Two popular methods include the scatter/gather approach, and the probabilistic EM-algorithm. The co-clustering method is also commonly used for text data. Topic modeling can be viewed as a probabilistic modeling approach that shares characteristics of both dimensionality reduction and clustering. The problem of novelty detection is closely related to text clustering. Streaming text clustering algorithms can be used for novelty detection. Data points that do not fit in any cluster are reported as novelties.

Among the classification methods, decision trees are not particularly popular for text data. On the other hand, instance-based methods, Bayes methods, and SVM methods are used more commonly. Instance-based methods need to be modified to account for the noise effects of synonymy and polysemy. The multinomial Bayes model is particularly popular for text classification of long documents. Finally, the *SVMPerf* method is commonly used for efficient text classification with support vector machines.

13.8 Bibliographic Notes

An excellent book on text mining may be found in [377]. This book covers both information retrieval and mining problems. Therefore, issues such as preprocessing and similarity computation are covered well by this book. Detailed surveys on text mining may be found in [31]. Discussions of the tree matching algorithm may be found in [357, 542].

The scatter/gather approach discussed in this chapter was proposed in [168]. The importance of projecting out infrequent words for efficient document clustering was discussed in [452]. The *PLSA* discussion is adopted from the paper by Hofmann [271]. The *LDA* method is a further generalization, proposed in [98]. A survey on topic modeling may be found in [99]. Co-clustering methods for text were discussed in [171, 172, 437]. The co-clustering problem is also studied more generally as biclustering in the context of biological data. A general survey on biclustering methods may be found in [374]. General surveys on text clustering may be found in [31, 32].

The text classification problem has been explored extensively in the literature. The *LSA* approach was discussed in [184]. Centroid-based text classification was discussed in [249]. A detailed description of different variations of the Bayes model in may be found in [31, 33].

The *SVMPerf* and *SVMLight* classifiers were described in [291] and [292], respectively. A survey on SVM classification may be found in [124]. General surveys on text classification may be found in [31, 33, 453].

The first-story detection problem was first proposed in the context of the *topic detection and tracking* effort [557]. The micro-cluster-based novelty detection method described in this chapter was adapted from [48]. Probabilistic models for novelty detection may be found in [545]. A general discussion on the topic of first-story detection may be found in [5].

13.9 Exercises

1. Implement a computer program that parses a set of text, and converts it to the vector space representation. Use tf-idf normalization. Download a list of stop words from `http://www.ranks.nl/resources/stopwords.html` and remove them from the document, before creating the vector space representation.

2. Discuss the weaknesses of the k-medoids algorithm when applied to text data.

3. Suppose you paired the shared nearest neighbor similarity function (see Chap. 2) with cosine similarity to implement the k-means clustering algorithm for text. What is its advantage over the direct use of cosine similarity?

4. Design a combination of hierarchical and k-means algorithms in which merging operations are interleaved with the assignment operations. Discuss its advantages and disadvantages with respect to the *scatter/gather* clustering algorithm in which merging strictly precedes assignment.

5. Suppose that you have a large collection of short tweets from Twitter. Design a Bayes classifier which uses the identity as well as the exact position of each of the first ten words in the tweet to perform classification. How would you handle tweets containing less than ten words?

6. Design a modification of single-linkage text clustering algorithms, which is able to avoid excessive chaining.

7. Discuss why the multinomial Bayes classification model works better on longer documents with large lexicons than the Bernoulli Bayes model.

8. Suppose that you have class labels associated with documents. Describe a simple supervised dimensionality reduction approach that uses *PLSA* on a derivative of the document-term matrix to yield basis vectors which are each biased towards one or more of the classes. You should be able to control the level of supervision with a parameter λ.

9. Design an EM algorithm for clustering text data, in which the documents are generated from the multinomial distribution instead of the Bernoulli distribution. Under what scenarios would you prefer this clustering algorithm over the Bernoulli model?

10. For the case of binary classes, show that the Rocchio method defines a linear decision boundary. How would you characterize the decision boundary in the multiclass case?

11. Design a method which uses the EM algorithm to discover outlier documents.